「Icarus Verilog」＋「IVI」の使い勝手を試す
――波形表示GUI付きのフリー・シミュレータ

デバイスの記事　関連データ

竹本 悟

ここでは，フリー・ソフトウェアとして提供されている波形表示ソフトウェア「IVI」を紹介する．Icarus VerilogなどのHDLシミュレータにIVIを組み込むことにより，波形表示GUI付きのシミュレータとして利用できる．（編集部）

フリーのVerilog HDLシミュレータとして知られている「Icarus Verilog」注1はGUI（graphical user interface）を備えていないので，シミュレーションを行ってもそのままでは波形を観測できません．従来はシミュレーション結果をvcdファイルとして記録し，それをほかの波形ビューワ（GTKWaveなど）で見るという手順を踏んでいました．

しかし，最近では，波形を表示するGUIソフトウェア「IVI」の開発プロジェクトが進められています．これを利用すれば，Icarus Verilogのシミュレーション結果を簡単に波形表示できます．また，条件を変更して再シミュレーションするといった，回路のデバッグに必要な制御をGUIで行えるようになります．

IVIはIcarus Verilogとはまったく独立に開発された，シミュレータ用のGUIです．本来はvcdファイルの波形ビューワだったようで，IVIそのものはシミュレーション機能を持っていません．シミュレータを組み込むことにより，統合された実行環境として働きます注2．

IVIの開発プロジェクト（URLは「http://ivi.sourceforge.net/」）は，Icarus Verilogへのインターフェースを開発し，またIcarus VerilogをIVIに組み込んで，Webサイトで提供しています注3．

● ツールをインストールする

それではさっそく，シミュレータ（Icarus Verilog）を組み込んだIVI（以下，IVIと表記）をWebサイト（http://sourceforge.net/project/showfiles.php?group_id=53425）から入手してみましょう注4．なお，Icarus Verilogを組み込んだIVIとオリジナル版のIcarus Verilogは共存できません．すでにIcarus Verilogをパソコンにインストールしている場合には，あらかじめ削除しておく必要があります．

IVIはWindowsやLinux，BSD，Solarisなど，さまざまなOSに対応しています．筆者はLinux版とWindows版（95，98，2000，XPに対応）を試してみました．

Windows版のインストールは非常に簡単です．「ivi-0.4-pre-20031121.setup.exe」というファイルをダウンロードしてダブルクリックすれば，表示される指示に従うだけでインストールが完了します．

Linux版をインストールする場合には，ソース・コードからコンパイルする必要があります．詳しくはp.98のコラム「Linux版のインストール」を参照してください．

● シミュレーションを実行する

シミュレーションを実行するためには，まずVerilog HDLのソース・コード（*.v）をコンパイルし，シミュレーション用のオブジェクト（*.vvp）を生成する必要があります．

1）Verilog HDLソース・コードのコンパイル

コンパイルにはiverilogコマンドを使用します．iverilog

注1：Icarus Verilogは，Stephen Williams氏が本ツールを作成し，フリー・ソフトウェアとして提供している．Verilog 1995，Verilog 2001，SystemVerilogに対応している．
注2：対応しているシミュレータはVerilog HDLシミュレータ「cvar」と「Icarus Verilog」である．また，今後，VHDLシミュレータ「GHDL」にも対応する予定だそうだ．
注3：IVIの開発プロジェクトは，現在，IVIをEclipseベースに移行しつつある．2005年3月現在，EclipseベースのIVIのバージョンが1.0.x，それ以前のIVIのバージョンが0.xという扱いになっている．Icarus Verilogを組み込んだIVIは0.xのほうにあたる．
注4：IcarusのWebサイト（http://icarus.com/eda/verilog/）からダウンロードできるのはIcarus Verilogのオリジナル版のみである．

コマンドは，IVIのコマンド・ウィンドウからでもコマンド・プロンプト（ターミナル）からでも実行できます．iverilogコマンドの使用例を以下に示します注5．

```
iverilog［ソース・コード名］-o［出力ファイル名］-s
                              ［トップ・モジュール名］
```

生成するオブジェクトのファイル名はデフォルトでは「a.out」になりますが，IVIを利用するためには拡張子をvvpにしておく必要があります．そこで，oオプションを利用して出力するファイル名を指定します．また，sオプションを利用してトップ・モジュールを指定します（指定しなくても自動的に探しに行ってくれるようだが，作者は指定することを推奨している）．

コンパイラが出力するエラー・メッセージは，まあ普通といったところです．ソース・コードのエラーのある場所を行数で示してくれます．商用のシミュレータに比べてやや不親切なようにも思いますが，使ううえではそれほどの不つごうは感じませんでした．

2）信号の選択とシミュレーションの実行

無事にコンパイルが完了したら，生成したオブジェクトをツール上で開きます．まだツールを起動していない場合，以下のコマンドを入力してIVIを起動します．

```
% $IVI_HOME/bin/ivi
```

IVIが起動したら（**図1**），メニュー・バーから「File」→「Open」→「Design」を選択してファイル（*.vvp）を指定します注6．次に，シミュレーションを開始する前に表示する信号を指定します．メニュー・バーから「File」→「New」→「Waveform Window」で波形表示ウィンドウを開きます．

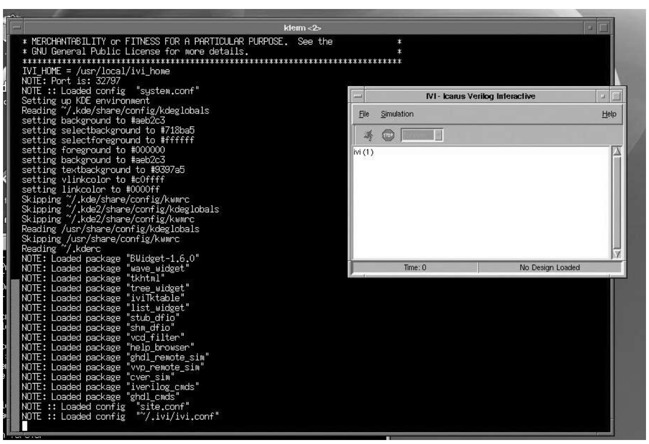


図1　IVIを起動したところ

注5：iverilogのコマンド・オプションについては，Linux版なら「man iverilog」で確認できる．Windows版なら，IVIのコマンド・ウィンドウから「iverilog -h」と入力することで確認できる（ただしオプションの種類のみで，詳細な説明はない）．これらのオプションは，米国Cadence Design Systems社の論理シミュレータ「Verilog-XL」のオプションにそっくりである．

注6：または，コマンド・ウィンドウで「load_design［ファイル名］」を実行する．

COLUMN　Linux版のインストール

まず，下記の二つのファイル注Aをダウンロードします．

- ivi-home-tcl-0.4-20031121.tar.gz
- ivi-verilog-20031009.prepatched.tar.gz

前者はGTK（The GIMP Toolkit）関連のパッケージ，後者はIcarus VerilogとIVIのプログラム本体です．ダウンロードしたら，以下の手順でGTKのライブラリをコンパイルします．エラーが出ずに完了したら，作成したディレクトリ（以下の例ではivi-built）の中にivi_homeというディレクトリができているのを確認してください．

```
% tar xvfz ivi-home-tcl-0.4-20031121.tar.gz
% mkdir  ivi-built   ← ディレクトリ名は何でもよい
% cd ivi-built
% ../ivi-home-tcl-0.4-20031121/configure
% make
```

次に，IVI本体をコンパイルします．

```
% cd ..
% tar xvfz ivi-verilog-20031009.prepatched.tar.gz
% mkdir bld_ivi-0.4
% cd bld_ivi-0.4
% cp -r ../ivi-home-tcl-0.4-20031121/ivi_home .
% ../ivi-verilog-20031009.prepatched/configure
% make
% make install   ← root権限で行う
```

コンパイルが無事に終了したら，ivi_homeを適当なディレクトリの下にコピーしてください．筆者は/usr/localの下にコピーしました．

なお，一般的なフリー・ソフトウェアでは，解凍してできたディレクトリの中でconfigureを実行しますが，このソフトウェアの場合，別のディレクトリ（bld_ivi-0.4）を作る必要があります．それをしないでいきなりconfigureを実行すると，makeするときにエラーが発生してしまいます．

注A：この二つのファイルが，本稿執筆時点での最新版である．ファイル名の後半の番号はバージョンを示しているので，ダウンロードするときはよく確認していただきたい．

そこでStructureタブを選択すると，ソース・コードで使用されている信号がツリー状に表示されます(**図2**)．確認したい信号をマウスでクリックしてハイライトさせ，マウスの右ボタンをクリックしてメニューから表示条件を選択します．そして，Waveformタブを選択し，［Run simulation］ボタン(走る人間を示したアイコン)をクリックするとシミュレーションを実行します注7．

GUIからシミュレーションを実行する場合には時間指定ができないので，かならずソース・コードに`$finish`を入れておくようにしましょう．そうしないと無限に実行されてしまいます．

なお，IVIのコマンド・ウィンドウから使用できるコマンドについては，p.103のコラム「IVIのコマンド」を参照してください．コマンドの種類は商用のシミュレータよりシンプルで，理解しやすいものでした．また，コマンドがわからない場合には，IVIに付いているヘルプ(英文)を参照するとよいでしょう．

● サンプル回路をシミュレーションしてみる

IVIにはサンプルのVerilog HDL記述が付属しています．何でもよいからシミュレーションしてみたい，という方は，このコードでテストしてみてください．

サンプル・コードはexamplesフォルダ内にあります．例えば，この中のverilog\miniriscフォルダにあるのは，簡単なRISCプロセッサ・コアのサンプルです．なお，Linux版に付いているdoファイルは少し修正が必要です注8．

1）Linux版の場合

Linux版の場合，ファイルを修正済みのものに置き換えた後，以下のようにしてサンプル・コードをビルドします．エラーが出ずに終了すると，このディレクトリにrisc_core.vvpというファイルができています．

```
% cp -r $IVI_HOME/example/verilog
                          /minirisc minirisc
% cd minirisc/src
% ../build
```

次に，IVIを起動してdoファイルを読み込みます(メニュー・バーから「File」→「Open」→「Command File」を選択して，doファイルを開く)．doファイルはIVIの起動時に指定することもできます．起動時に指定するには，

```
% $IVI_HOME/bin/ivi -do minirisc_icarus
                               _runtest.do
```

と入力します．doファイルを読み込むとシミュレーションが実行され，波形が表示されます(**図3**)．

2）Windows版の場合

Windows版の場合は，あらかじめ用意されているバッチ・ファイル(doファイル)を開くだけでrisc_core.vvpが

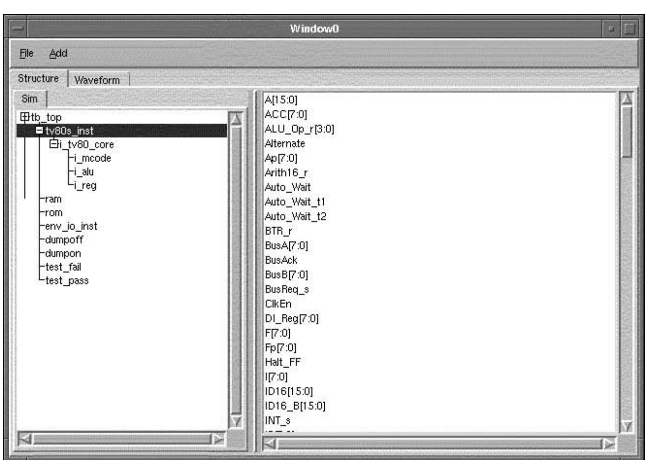


図2　波形表示させたい信号を選択する
波形表示ウィンドウを開いてからStructureタブを選択し，信号を指定する．



図3　サンプル回路(minirisc)のdoファイルを実行したところ
minirisc_icarus_runtest.doを読み込むとシミュレーションが実行され，波形ウィンドウが自動的に開いて波形を表示する．

注7：コマンドで指示する場合には，「wave add［信号名］」などを実行する．後は，runコマンドを打ち込めば，シミュレーションが実行される．
注8：修正済みのファイルは本誌のWebサイト(http://www.cqpub.co.jp/dwm/)から入手できる．

生成され，シミュレーションを実行できます．具体的には，メニュー・バーから「File」→「Open」→「Command File」を選択し，examples \ verilog \ minirisc フォルダ内のminirisc_icarus_runtest.doを開きます[注9]．doファイルはシミュレーション用（minirisc_icarus_runtest.do）と波形表示用（minirisc_icarus_wave.do）に分かれていますが，シミュレーション用のdoファイルを実行すれば，その中で波形表示のdoファイルも呼び出します．

● 自作のコードでシミュレーションしてみる

サンプルを動かすだけではおもしろくないので，自作のコードもいくつか試してみました．コードを自作する場合，IVIにはソース・コード・デバッガの機能はないので，手ごろなテキスト・エディタと組み合わせて使うことになります．なお，デバッグ時など，シミュレーションを繰り返す際には，コマンド・ウィンドウからいちいち命令を入れるのはめんどうなので，doファイルを作っておくことをお勧めします．

1）波形と文字を表示する

まずは`initial`文を使って，波形と文字変数を表示してみました（**リスト1**，**図4**）．ソフトウェアのテスト・プログラムで文字を表示する場合，「Hello World！」と書かせるプログラムをよく見かけますが，ハードウェアのシミュレータらしい文章にしました．

2）signedとunsignedの動作をテストする

Verilog 2001からの仕様である`signed`と`unsigned`の動作をテストしてみました（**リスト2**）．Verilog HDLの記述はややわかりにくいのですが，機能としては問題ないようです．

3）アップダウン・カウンタのシミュレーション

ちょっと実用的な機能として，アップダウン・カウンタを作ってテストしてみました（**リスト3**）．Verilog 2001からの仕様である，パラメータの外部定義の機能を試してみましたが，正しく動作しているようです．ここでは，一つのカウンタの記述で2種類のカウンタを実現しています．

4）Z80のシミュレーション

自作のコードではないのですが，Z80互換のプロセッサ・コアのシミュレーションを試してみました．Z80のフリーのコードとしてはVHDLで記述された「T80」が有名ですが，Verilog HDL版である「TV80」もリリースされているので，こちらのコードでシミュレーションしてみることにしました．TV80は，T80と同じようにOPENCORESのWebサイト（URLは「http://www.opencores.com/」）からダウンロードできます．

結果は，非常に満足のいくものでした．Verilog HDLのコードをほとんどいじらなくてもシミュレーションを実行

リスト1　波形と文字を表示する

```
`timescale 1 ns/10 ps
  module  novice;
   reg  sig1;
   reg  [21*8:1]  str1="My first simuration!!";
   initial  begin
    $display("%s",str1);
          sig1=0;
    #10  sig1=1;
    #10  sig1=0;
    #10  $finish;
   end
  endmodule
```

（**a**）Verilog HDLソース・コード（notice.v）

```
iverilog -s novice -o _work/novice.vvp novice.v
load_design  _work/novice.vvp
wave add sig1
wave add str1 -radix str
run forever
wave zoom full
```

（**b**）doファイル（notice.do）

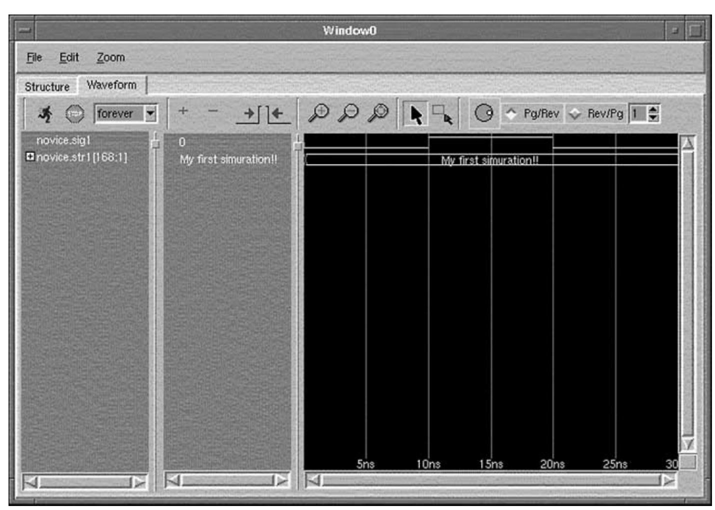


図4　波形と文字を表示するプログラムの実行結果
リスト1（a）のソース・コードに対し，リスト（b）のdoファイルを実行した結果の波形．

注9：ただし，doファイルを開く前に，あらかじめIVIのコマンド・ウィンドウで「cd examples/verilog/minirisc」などのコマンドを入力し，現在のディレクトリ位置をexamples \ verilog \ miniriscまで移動しておく必要がある．

できました(**リスト4**,**図5**).ソフトウェアとしては,ごく簡単なプログラムをROMにロードさせて実行してみました.

● 論理合成とSystemVerilogへの対応について

前項の**4)**で使用したTV80のソース・コードを使って,Icarus Verilogの論理合成機能を試してみました.論理合成はiverilogコマンドのSオプションで行い,EDIF形式のネットリストが生成できるようです.しかし筆者が試した範囲では,記述に問題があるというエラーが出て,うまくいきませんでした.`if`文の解釈がうまくいかないようなのです.このコードはすでにASICとして実績があるものなので,コード側の問題とは考えにくいのですが….Icarus Verilogの合成時のコードの解釈のしかたに相違があるように思われます.エラーが出ているのは一部のコードだけなので,その部分の記述のしかたを修正すれば合成できるようになるものと思われます.おそらく,iverilogの合成のくせをつかめばいいのでしょうが,それに関するドキュメントは存在せず,今のところお手上げの状態です.

Icarus VerilogはVerilog 1995,Verilog 2001,System Verilogの三つのバージョンに対応しているので,System Verilogのテストもしてみようと思いましたが,手ごろなSystemVerilogのサンプル・コードが入手できず,断念しました.

なお,バージョンの選択はiverilogコマンドでコンパイルするときにオプションとして指定します.デフォルトの状態ではSystemVerilogが選択されます.SystemVerilogはVerilog 2001のスーパセットであり,Verilog 2001はVerilog 1995とほとんど上位互換なので,わざわざバージ

リスト2 signedとunsignedの動作をテストする

```
`timescale 1 ns/10 ps
 module  novice_a;
  reg  [3:0] int1,int2,int3;
  reg signed [3:0] int1s,int2s,int3s;

  initial  begin
   int1=10;
   int2=-2;
   int3=7.5;
   int1s=10;
   int2s=-2;
   int3s=int3;
   $display("int1=%b int2=%b int3=%b",int1,int2,int3);
   $display("int1s=%b int2s=%b int3s=%b",int1s,int2s,int3s);
   $display("int1=%d int2=%d int3=%d",int1,int2,int3);
   $display("int1s=%d int2s=%d int3s=%d",int1s,int2s,int3s);


   $finish;
  end
 endmodule
```

(**a**) Verilog HDLソース・コード(novice_a.v)

```
iverilog -s novice_a -o _work/novice_a.vvp novice_a.v
load_design  _work/novice_a.vvp
run forever
```

(**b**) doファイル(novice_a.do)

リスト3 アップダウン・カウンタ

```
//parametarized up-down counter
module p_upd    (out, up_down, clk,reset);
    parameter bcnt=3;
    output [bcnt:0] out;
    input up_down, clk, reset;
    reg [bcnt:0] out;
    always @(posedge clk)
    if (reset) begin // active high reset
        out <= 0 ;
    end
    else if (up_down) begin
        out <= out + 1;
    end
   else begin
        out <= out - 1;
    end
endmodule
```

(**a**) Verilog HDLソース・コード(p_upd.v)

```
`timescale 1 ns/ 10 ps
module testp_upd;
parameter bitc1=5,bitc2=3;
reg clk,upd,reset;
wire [bitc1:0] cout1;
wire [bitc2:0] cout2;
initial begin
  reset=0;
  upd=1;
  #51 reset=1;
  #60 reset=0;
  #200 upd=0;
  #300 $finish;
end
initial begin
  clk=0;
  forever #5 clk=~clk;
end
defparam u1.bcnt=bitc1;
defparam u2.bcnt=bitc2;
p_upd u1(cout1,upd,clk,reset);
p_upd u2(cout2,upd,clk,reset);

endmodule
```

(**b**) Verilog HDLソース・コード(testp_upd.v)

```
iverilog -s testp_upd -o _work/testp_upd.vvp
                                  testp_upd.v p_upd.v
load_design  _work/testp_upd.vvp
wave add clk
wave add upd
wave add reset
wave add cout1 -radix dec
wave add cout2 -radix dec
run forever
wave zoom full
```

(**c**) doファイル(testp_upd.do)

リスト4　Z80のシミュレーション（一部）

```
`timescale 1ns/10ps
module tb_top;

  reg         clk;
  reg         reset_n;
  reg         wait_n;
  reg         int_n;
  reg         nmi_n;
  reg         busrq_n;
  wire        m1_n;
  wire        mreq_n;
  wire        iorq_n;
  wire        rd_n;
  wire        wr_n;
  wire        rfsh_n;
  wire        halt_n;
  wire        busak_n;
  wire [15:0] A;
  wire [7:0]  di;
  wire [7:0]  do;
  wire        ram_rd_cs, ram_wr_cs, rom_rd_cs;

  always
    begin
      clk = 1;
      #50;
      clk = 0;
      #50;
    end

  assign rom_rd_cs = !mreq_n & !rd_n & !A[15];
  assign ram_rd_cs = !mreq_n & !rd_n & A[15];
  assign ram_wr_cs = !mreq_n & !wr_n & A[15];

  tv80s tv80s_inst
    (
     // Outputs
     .m1_n                          (m1_n),
     .mreq_n                        (mreq_n),
     .iorq_n                        (iorq_n),
     .rd_n                          (rd_n),
     .wr_n                          (wr_n),
     .rfsh_n                        (rfsh_n),
     .halt_n                        (halt_n),
     .busak_n                       (busak_n),
     .A                             (A[15:0]),
     .do                            (do[7:0]),
     // Inputs
     .reset_n                       (reset_n),
     .clk                           (clk),
     .wait_n                        (wait_n),
     .int_n                         (int_n),
     .nmi_n                         (nmi_n),
     .busrq_n                       (busrq_n),
     .di                            (di[7:0]));

  async_mem ram
    (
     // Outputs
     .rd_data                       (di),
     // Inputs
     .wr_clk                        (clk),
     .wr_data                       (do),
     .wr_cs                         (ram_wr_cs),
     .addr                          (A[14:0]),
     .rd_cs                         (ram_rd_cs));

  async_mem rom
    (
     // Outputs
     .rd_data                       (di),
     // Inputs
     .wr_clk                        (),
     .wr_data                       (),
     .wr_cs                         (1'b0),
     .addr                          (A[14:0]),
     .rd_cs                         (rom_rd_cs));

  env_io env_io_inst
    (
     // Outputs
     .DI                            (di[7:0]),
     // Inputs
     .clk                           (clk),
     .iorq_n                        (iorq_n),
     .rd_n                          (rd_n),
     .wr_n                          (wr_n),
     .addr                          (A[7:0]),
     .DO                            (do[7:0]));

  initial
    begin
      reset_n = 0;
      wait_n = 1;
      int_n  = 1;
      nmi_n  = 1;
      busrq_n = 1;
      $readmemb ("test.dat",  tb_top.rom.mem);
//modified by S.Takemoto
      repeat (20) @(negedge clk);
      reset_n = 1;
    end

`include "env_tasks.v"

endmodule // tb_top
```

（**a**）Verilog HDLソース・コード（tb_top.v）

```
iverilog -I env  -s tb_top -o tb_top.vvp  -v
   env/tb_top.v env/env_io.v env/async_mem.v
   rtl/core/tv80_alu.v rtl/core/tv80_mcode.v
    rtl/core/tv80_reg.v rtl/core/tv80_core.v
                         rtl/core/tv80s.v
load_design tb_top.vvp
wave add tb_top.*
run 4us
wave zoom full
```

（**b**）doファイル（tb_top.do）

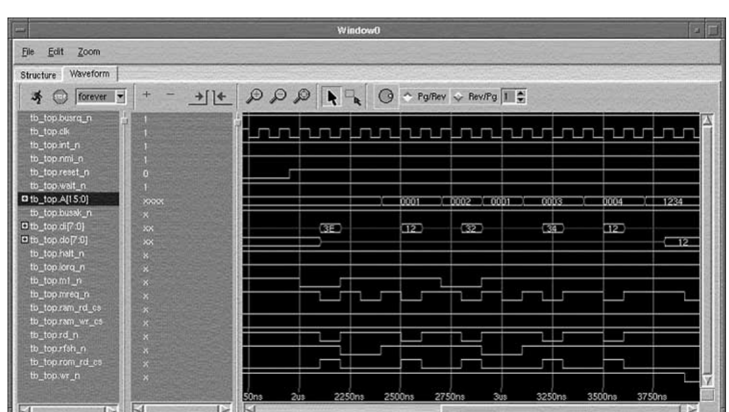


図5
Z80のシミュレーション結果
リスト1（a）のソース・コードやそのほかのTV80のソース・コードに対し，リスト1（b）のdoファイルを実行した結果の波形．

ョンを指定してシミュレーションする必要はなさそうです．

● 動作感は快適

全体の感触として，シミュレータとしては非常によくできているというのが筆者の実感です．Icarus Verilogが昔からあるVerilog HDLのシミュレータであり，GNUのEDAプロジェクトGEDAの一翼を担っていたので，シミュレータとしてこなれているのだと思われます．IVIとの結合もよくできています．プログラムの起動も驚くほどスムーズです．詳しく測ったわけではありませんが，商用のシミュレータと比較してもひけをとらないように思われます．

表示する信号の指定は，コマンド・ウィンドウから指定する方法と，波形ウィンドウの構造ツリーからマウスでクリックする方法があります．どちらもそれほどむずかしくありません．

便利だと思ったのはスクロール・ダイヤル機能です（**図3**）．波形表示ウィンドウにある小さなダイヤルをマウスで回すと，それに合わせてスクロールします．矢印をクリックするよりも快適に操作できました．

● 課題が残るASICライブラリへの対応

Verilog HDLのユーザはFPGAよりもASICの設計に関係している方が多いと思います．ASIC開発で使用するには，ASICベンダが供給するASICライブラリをシミュレーションできなければなりません．

多くのASICライブラリはVerilog HDLのUDP（ユーザ定義プリミティブ）で記述されています．また，遅延条件は`specify`ブロック内の`specparameter`で記述されています．Icarus VerilogでASICの開発を行うためにはこのASICライブラリが使えなければなりません．`specify`ブロックはIcarus Verilogでサポートされているようなので，シミュレータとしての機能には問題はないでしょう．ただし，Icarus VerilogをASIC開発に使用するには，現在のところ二つの問題点があります．

一つは，ASICベンダがライブラリのソース・コードを公開していないことです．Icarus Verilogはライブラリの

COLUMN　IVIのコマンド

IVIのコマンド・ウィンドウから与えることができるコマンドは四つのカテゴリに分かれています．

1）シミュレーション・コマンド

load_design［vvpファイル名］

iverilogでコンパイルされたファイルをロードします．

run［時間］¦forever

シミュレーションを指定した時間で実行します．時間の代わりに`forever`と指定すると，ソースの中に`$finish`が出てくるまで実行し続けます．

2）テストベンチ用のコマンド

筆者がテストした限りでは，テストベンチ用の二つのコマンドは現在のところ機能していないようです．入力するとハングアップしてしまいました．

get value［信号名］

現在のシミュレーション時間における信号の値を表示します．

put value［信号名］［値］

信号に強制的に値を与えます．

3）波形表示用のコマンド

wave lookup

波形表示ウィンドウの名まえを返します．

wave add［信号名］

指定された信号を波形表示ウィンドウに表示します．さまざまなオプションがありますが，詳細はIVIに付いているヘルプを参照してください．

wave add_cusor

カーソルを追加します．

wave zoom［種別］¦［開始時間］［終了時間］

表示を拡大/縮小します．種別としては，in（縮小），out（拡大），full（全面表示），last（zoomコマンドを実行する前の状態に戻す）があります．

4）波形データベースのコマンド

IVIはシミュレーション結果（波形データ，形式は*.sdbまたは*.vcd）を保存したり，読み込んだりできます．

sdb open［ファイル名］-type［タイプ名］

保存してあるシミュレーション結果を読み込みます．vcd形式の場合には，タイプ名をvcdと記述します．

sdb save［ファイル名］-type［タイプ名］

シミュレーション結果を保存します．

ソース・コードがないとシミュレーションできないのです．もう一つは，SDFのバック・アノテーションにまだバグがあり，完全に動作しないことです．SDFのバック・アノテーションはVerilog 2001からシステム関数`$sdf_backanote`として定義されており，Icarus Verilogではこの`$sdf_backanote`をサポートすることによってSDFのバック・アノテーションを実現しようとしています．これは，現在プラグインとして提供されている状態で，まだソース・コードの中に組み込まれていないようです[注10]．

この問題はIcarus Verilogの改良を待つほかはありませんが，順次対応されるようです．ただ，ここにある記述を読む限り，現時点では使いものにならないようです．もちろん，遅延を問題にしなければシミュレーションはできるようですが，それではあまりネットリストでシミュレーションする意味がないでしょう．

● FPGA設計なら実用性あり，後は今後に期待

実用になるか？ と言えば，Yesでしょう．エラー表示が少し不親切など，いくつか不満はありますが，FPGAを設計する際には十分に使えると思います．ASIC設計については，SDFの問題が解決すれば実用になると思いますし，ネットリストのバック・アノテーションを除けば機能するでしょう．論理合成の機能は，いまひとつと言ったところでしょうか．これは将来に期待するしかありません．

現在の改善の課題は，多分にIcarus Verilog側にあります．作者が言っているように，Icarus Verilogはまだ発展途上にあるようです．これからさまざまな改良が予定されているようなので，非常に期待されるところです（ただし，あくまでもフリー・ソフトウェアなので，「早く改良しろ」などと要求しないでおいていただきたい．開発者の方がボランティアでやってくださっていることをお忘れなく…）．

たけもと・さとる
（有）サイバーラボ

竹本 悟．1952年神戸市生まれ．高校時代にラジオやアンプを作っていた世代ですが，いつの間にかHDL設計を飯の種にするようになっていました．サラリーマンを20年，二つの会社に勤めましたが，やっぱりやってみたくて独立しました．ときどきCQ出版でもセミナの講師を務めています．趣味は碁，つり，キャンプなどいろいろやりますが，回路を考えているときがいちばん楽しいと思っています．もちろん熱烈な阪神ファンです．

注10：http://icarus.com/eda/verilog/plug-ins.htmlの「SDF Support」を参照のこと．